

### 5.1.2 Linkkomponenten

Das Grundelement des Web sind seine Hyperlinks. Zu Beginn waren das simple Verknüpfungen zwischen unterschiedlichen Dokumenten, die beliebig weit verteilt sein konnten. Solche statischen Links lassen sich geradezu trivial in Wicket verwenden: Man schreibt einfach wie gewohnt `<a href="linkziel">Linktext</a>` in das HTML-Template. Heutzutage sind Links natürlich immer noch hauptsächlich »Dokumentenzeiger«, aber sie werden nun auch zur Steuerung von Webanwendungen benutzt. Sie können sowohl auf dem Server als auch – mittels JavaScript<sup>2</sup> – auf dem Client beliebige Aktionen auslösen.

#### Links innerhalb einer Anwendung

Um dem Benutzer Navigationsmöglichkeiten innerhalb der Anwendung zu bieten, gibt es z. B. `Link`. Ein Beispiel für einen solchen Link enthält das `SearchUserPanel`.

#### Listing 5.4

*HTML und Java für die Klasse `SearchUserPanel` (Auszug) – ein Beispiel für Seitennavigation*

```
<a wicket:id="createUser" href="#">
  Neuen Benutzer anlegen
</a>
```

```
border.add(new Link("createUser") {
  @Override
  public void onClick() {
    setResponsePage(CreateUserPage.class);
  }
});
```

Listing 5.4 zeigt, dass der `Link` sich – wie alle Linkkomponenten – an ein `<a>`-Tag anhängt. Sobald der Benutzer auf diesen Link klickt, wird `setResponsePage(Class<Page>)` aufgerufen und die angegebene Seite per Defaultkonstruktor erzeugt und gerendert. Wenn die Zielseite keinen Defaultkonstruktor besitzt (Wicket kann die Seite also nicht automatisch erzeugen) oder wenn eine schon bestehende Instanz verwendet werden soll, würde stattdessen `setResponsePage(Page)` aufgerufen. Da `onClick` erst beim Klick aufgerufen wird, wird die Zielseite auch erst dann erzeugt und belegt so nicht unnötig Platz auf dem Server.

Die tatsächlich im Browser anklickbare Fläche entspricht dabei genau dem, was innerhalb des `<a>`-Tags steht. Im Beispiel ist das einfach ein Text, es können aber auch beliebige Wicket-Komponenten verwendet werden, z. B. ein `Label`, um den Text dynamisch anzupassen. Diese

<sup>2</sup>Siehe Kapitel 8.

Komponenten müssen natürlich dem Link über `add()` hinzugefügt werden.

Link erzeugt eine relativ unansehnliche URL. Sie sieht ungefähr aus wie `http://localhost:8080/wicket-wbs-web/?wicket:interface=:4:middleColumn:listPanel:2:createUser::ILinkListener::`, die genauen Werte hängen vom momentanen Zustand der Session ab. Für Links, die ohnehin nur innerhalb der Anwendung gelten, ist das aber akzeptabel. Links in Wicket haben aber noch eine weitere Eigenschaft: Sie gelten nur *innerhalb der aktiven Session*.

*Sessionrelative Links*

Wie schon das Escaping von Sonderzeichen im Label, so hat auch dieser Sessionbezug der Links Vor- und Nachteile, wobei meistens die Vorteile überwiegen, da es auch hier wieder um die Sicherheit der Anwendung geht. Eine in den letzten Jahren aufgekommene Attacke basiert darauf, einem Benutzer auf einer fremden Website einen Link in die Anwendung unterzuschleusen, der dann in der Anwendung eine Aktion mit den Rechten dieses Benutzers auslösen soll. Das funktioniert natürlich nur, wenn der Angreifer vorhersagen kann, welche URL zu welcher Aktion gehören wird. In Wicket ist dieser Angriff nahezu unmöglich geworden: Die generierten URLs sind relativ zur Session und enthalten z. B. die Position der aktuellen Seite in der Page Map sowie die Versionsnummer, die sich bei jedem Aufruf der Seite erhöht. Dadurch ist es sehr schwierig, eine gültige Link-URL vorherzusagen.

Sowohl die URLs als auch der Sessionbezug der Links lassen sich beeinflussen. Zum Sessionbezug gibt es z. B. `BookmarkablePageLinks`, zur Anpassung der URLs erfahren Sie mehr in Abschnitt 11.5 auf Seite 295.

### Stabile Links mit `BookmarkablePageLink`

Mit `BookmarkablePageLink` statt `Link` wird die URL des Links »stabil« und kann, wie der Name schon sagt, vom Anwender auch als Bookmark gespeichert und jederzeit später wieder aufgerufen werden. Das funktioniert nur, wenn die Seitenklasse statt einer fertigen Instanz verwendet wird. Da die Session nicht zur Verfügung steht, gibt es auch keine Link-Komponente, deren `onClick` aufgerufen werden könnte.

Trotzdem können der verlinkten Seite noch Parameter übergeben werden, sie müssen nur unabhängig von der Session sein. Statt des Defaultkonstruktors der Seite kann Wicket noch einen anderen aufrufen, der genau ein Argument vom Typ `PageParameters` enthält. Dabei handelt es sich um eine Map aus Schlüssel-Wert-Paaren, die die Parameter für die neue Seite darstellen. Da diese Werte später alle als Query-String an die URL angehängt werden, müssen sie ein brauchbares `toString` haben. Für die üblichen Typen wie `Integer`, `Boolean` usw. ist das bereits der Fall. In Listing 5.5, einem Auszug aus den Beispielkom-

*PageParameters*

ponenten `ProjectListPanel` und `ProjectDetailsPage`, ist die Verwendung von `BookmarkablePageLink` und `PageParameters` dargestellt.

**Listing 5.5**

*ProjectListPanel.java*  
und  
*ProjectDetailsPage.java*  
(Auszug) – Beispiel für  
*BookmarkablePage-  
Link*

```
ProjectListPanel.java:
...
PageParameters pageParameters =
    new PageParameters();
pageParameters.put("projectId", project.getId());
item.add(new BookmarkablePageLink<ProjectDetailsPage>(
    "tasks", ProjectDetailsPage.class,
    pageParameters).add(new Label("id")));
...

ProjectDetailsPage.java:
...
public ProjectDetailsPage(PageParameters
    pageParameters) {
    long projectId = pageParameters.getLong("projectId");
    Project project = projectManager.retrieve(projectId);
    ...
}
...
```

Vor der Generierung des Links wird ein neues `PageParameters`-Objekt erzeugt mit einem Parameter "projectId". Dann wird der `BookmarkablePageLink` aufgebaut: Er bekommt seine Wicket-ID, die Klasse der Zielseite und schließlich die `PageParameters`. Zuletzt wird ihm noch ein Label hinzugefügt, das den anklickbaren Linktext liefert. Wenn der Benutzer nun auf den Link klickt, wird der gezeigte Konstruktor aufgerufen. Das Argument "projectId" wird wieder aus den `PageParameters` herausgeholt – `getLong()` gehört zu einer Reihe von Hilfsfunktionen, die aus den Strings einfache Datentypen machen können – und verwendet.

### <wicket:link>

In Wicket sind grundsätzlich Template (HTML) und Logik (Java) strikt getrennt. Einige sehr einfache Dinge wären jedoch im Code so trivial, dass die Autoren zusätzlich die Möglichkeit eingebaut haben, sie auch komplett im Template zu halten. Dazu gehören lokalisierte Texte<sup>3</sup> wie auch ein Tag für die automatische Erstellung von `BookmarkablePageLinks`. Dies ist nützlich, wenn es z. B. ein fixes Menü von Seiten gibt und nicht alles auch noch in Java mitgepflegt werden soll. In diesem Fall kann man, wie in Listing 5.6 gezeigt, Links in

<sup>3</sup><wicket:message>, siehe Abschnitt 11.1.

`<wicket:link>`-Tags einbetten. Diese Links haben dann kein `wicket:id`-Attribut, denn das würde sie zu vollwertigen Komponenten machen, die auch im Code vorkommen müssten.

```
<wicket:link>
  <a href="sub/package/StartPage.html">
    Start
  </a><br/>
  <a href=" ../higher/package/SearchPage.html">
    Search
  </a><br/>
</wicket:link>
```

**Listing 5.6**

Verwendung von  
`<wicket:link>`

Die Linkpfade müssen zu den HTML-Templates in den Package-Verzeichnissen führen. Wicket generiert daraufhin entsprechende `BookmarkablePageLink`-Instanzen, die zu den jeweiligen Seitenklassen führen. Links auf die aktuelle Seite werden dabei automatisch deaktiviert dargestellt. So kann man den Aufwand einsparen, diesen recht statischen Teil der Anwendung dynamisch im Code aufzubauen.

**Automatische Links überall**

`<wicket:link>` dient dazu, die automatische Linkgenerierung nur für bestimmte Regionen im Template zu aktivieren. In der Standardeinstellung verändert Wicket keine Links außerhalb dieser Regionen. In der `init`-Methode der Anwendungsklasse kann dieses Verhalten mit `getMarkupSettings().setAutomaticLinking(boolean)` angepasst werden: `false` ist der Default, `true` sorgt dafür, dass Wicket *jeden* Link im Template bearbeitet, der kein eigenes `wicket:id`-Attribut hat.

`<wicket:link>` hat aber auch noch einen anderen Anwendungsbereich. Pfade zu Bildern, Stylesheets und Skripten sind in HTML üblicherweise relativ angegeben, und gerade in Webanwendungen wären absolute Pfade einem Deployment in einer anderen (z. B. Test-)Umgebung hinderlich. Je nachdem, wie die URL zur aktuellen Seite genau aussieht – abhängig davon, wie viele Schrägstriche enthalten sind<sup>4</sup> –, kann der Webbrowser einen relativen Pfad ganz unterschiedlich interpretieren. Durch das Einbetten in `<wicket:link>`-Tags kann man die Pfade zu diesen Ressourcen ebenfalls anpassen lassen. Ein Beispiel dafür sind die Sprach-Umschalt-Links im Template `Header.html` in Listing 5.7: Die Bilddateien `de.png` und `us.png` liegen im gleichen Package und damit im gleichen Verzeichnis wie das HTML-Template und die Header-Klasse. Daher braucht man hier kein Package-Präfix. Aber egal wie der Link

*Automatische Pfade  
für Bilder und Skripte*

<sup>4</sup>Das kann z. B. durch das Mounten von Seiten oder durch unterschiedliche `IRequestCodingStrategies` beeinflusst werden, siehe Abschnitt 11.5.

### Vorsicht mit Get-Requests

Get-Requests, also solche, die durch normale Links ausgelöst werden, sollten keine nicht zurücknehmbaren Datenveränderungen vornehmen. Es gibt die Geschichte einer mittelgroßen Webanwendung, die dadurch abgeschossen wurde, dass der Google-Bot über Nacht auf jeder Seite sorgfältig dem »Produkt löschen«-Link gefolgt ist.

Es gibt zwei Möglichkeiten, zu verhindern, dass Robots wie die von Suchmaschinen solche fatalen Aktionen auslösen, indem sie den Links folgen:

- Nur Post-Requests für solche Aktionen verwenden. Posts werden üblicherweise nur durch das Absenden von Formularen erzeugt.
- Die Aktion durch Ajax-Links auslösen (siehe Kapitel 8 auf Seite 215).

Außerdem sollten potenziell gefährliche Aktionen nur mit einer entsprechenden Berechtigung ausführbar sein. Wenn der Link nur nach einem Login funktioniert, stellt auch der Google-Bot keine Gefahr mehr dar.

Wie schon bei der Seitennavigation gesehen, muss für Link nur die Methode `onClick` implementiert werden. Im Gegensatz zu vielen früheren Frameworks muss man sich nicht um URLs, Request-Zyklen, Forwards und solche Dinge kümmern, nur um eine Aktion für den Benutzer auszuführen. Stattdessen kann man sich einfach darauf verlassen, dass die `onClick()`-Methode aufgerufen wird, wenn der Benutzer auf den Link klickt. Im Beispiel wird daraufhin der Logout durchgeführt, indem die Session gelöscht und der Benutzer auf die StartPage weitergeleitet wird.

Dabei ist zu beachten, dass Link immer erst einmal auf die aktuelle Seite zurückführt. Ohne den Aufruf von `setResponsePage()` fände *kein* Seitenwechsel statt, sondern die aktuelle Seite würde in ihrem aktuellen Zustand erneut ausgegeben. Ein solcher Link eignet sich also dazu, um aus klassischen GUIs bekannte Buttons und ähnliche Controls nachzuempfinden. Wenn eine andere Seite kommen soll, egal ob nur als Navigation oder auch als Ergebnisseite einer aufwendigen Serveraktion, geschieht das über `setResponsePage()`. Diese Methode hängt an `Component` und ist daher überall verfügbar, nicht nur in Links. Sie kommt in drei Varianten vor: `setResponsePage(Class<C extends Page>)`, `setResponsePage(Class<C extends Page>, PageParameters)` und `setResponsePage(Page)`.

Mit den Komponenten `AjaxLink` und `AjaxFallbackLink` ist es auch möglich, JavaScript- und Ajax-Funktionen aufzurufen. Mehr dazu in Kapitel 8.

*JavaScript/Ajax-Links*

aussieht, über den der Benutzer auf die aktuelle Seite gekommen ist, `<wicket:link>` sorgt dafür, dass der Browser immer den richtigen Pfad zu den Bildern angezeigt bekommt. Die Links selbst sind »normale« Wicket-Komponenten: `AutoLinkBookmarkablePageLinks` für Links aufseiten der Anwendung und `ExternalLinks` für absolute Links ins Netzwerk (Links, die eine vollständige URL enthalten). Links auf Ressourcen wie Bilder und Skripte werden als `ResourceReferenceAutoLinks` abgebildet. Programmatische Links und automatisch erstellte Links lassen sich problemlos kombinieren.

#### **Listing 5.7**

*Header.html (Auszug) –  
<wicket:link> für Bilder*

```
<wicket:link>
  <a href="#" wicket:id="langde">
    
  </a>
  <a href="#" wicket:id="langus">
    
  </a>
</wicket:link>
```

### **Links als Auslöser für Aktionen**

Wie eingangs erwähnt, können Links in modernen Webanwendungen nicht nur zur Navigation verwendet werden. Man sieht immer häufiger auch, dass Links wie Buttons in klassischen GUI-Anwendungen eingesetzt werden und eine Aktion auf dem Server auslösen (z. B. Teile der Oberfläche auf- oder zuklappen). Damit sollte man zwar ein bisschen vorsichtig sein – siehe den Kasten zu Get-Requests –, aber in vielen Fällen ist es nützlich, und Wicket bietet mit der Komponente `Link` hervorragende Unterstützung. Listing 5.8 zeigt das `AuthenticationStatePanel`, das einen Link zum Ausloggen aus der Anwendung enthält.

#### **Listing 5.8**

*AuthenticationState-  
Panel.java (Auszug) –  
Link und onClick*

```
add(new Link("logoutLink") {
  @Override public void onClick() {
    WBSSESSION.get().invalidate();
    setResponsePage(StartPage.class);
  }
});
```

### Links zum Erzeugen von Popup-Fenstern

Mit den vorgestellten Linkkomponenten ist es auch möglich, das Linkziel in einem neuen Fenster zu öffnen. Die einfachste Methode, dies zu erreichen, ist die Verwendung des Attributs `target` im Anchor-Element: `<a href=... target="newWindowId">...</a>`. Diese Methode scheitert allerdings, wenn man einen Button mithilfe von `<input type="button" .../>`, `<input type="submit" .../>` oder `<button>` verwenden möchte. Oft ist es auch gewünscht, mehr Kontrolle über das zu öffnende Browserfenster ausüben zu können, wie das Einstellen von Fenstergröße und Position oder das Ein-/Ausblenden des Browsermenüs und der Browser-Toolbar. In solchen Fällen ist man auf den Einsatz der JavaScript-Funktion `window.open()` angewiesen. Zum Öffnen eines Popup-Fensters über `window.open()` muss lediglich die Property `popupSettings` vom Typ `PopupSettings` auf dem Linkobjekt gesetzt werden. Listing 5.9 zeigt, wie man ein Popup-Window zur Anzeige von PDF parametrieren kann.

**Listing 5.9**  
Popup-Window-  
Einstellungen

```
WebResource projectsResource = ...

ResourceLink pdfDownload =
    new ResourceLink("pdfDownload", projectsResource);
PopupSettings popupSettings =
    new PopupSettings(PopupSettings.STATUS_BAR);
popupSettings.setWidth(500);
popupSettings.setHeight(700);
pdfDownload.setPopupSettings(popupSettings);
```

Die hier verwendete Linkkomponente (`ResourceLink`) hat die Aufgabe, einen Link auf eine Wicket-Ressource zu generieren und bei Aktivierung auszuliefern. Der Download-Mechanismus selbst wird detailliert in Abschnitt 11.2 auf Seite 281 beschrieben.

Über `PopupSettings` wird die Größe des Fensters (hier 500 x 700) festgelegt. Außerdem wird durch den Konstruktorparameter `PopupSettings.STATUS_BAR` die Browserstatusanzeige aktiviert. Die gewünschten Fensterdekorationsparameter müssen explizit eingeschaltet werden. Dies geschieht durch Oder-Verknüpfung der entsprechenden Konstanten: `new PopupSettings(PopupSettings.STATUS_BAR | PopupSettings.SCROLLBARS)`.

### 5.1.3 Formularkomponenten

Neben der Anzeige von Informationen ist die Verarbeitung von Benutzereingaben eine der Hauptaufgaben von webbasierten Systemen. Ty-